设计实验室 14　　6.01 –Fall 2011

我就是那张地图！

目标： 在这个实验室中，你将构建一个能够在机器人上运行的系统，使机器人能够在世界中移动时动态地构建周围障碍物的地图，并沿着动态规划的路径向目标前进。尽管传感器不可靠且障碍物不确定，但该系统仍能稳健运行。你将：

- 构建一个使用完美声纳信息的动态地图制作器

  并在 Soar 中用**软件实验室 14** 的规划器对其进行测试
- 改进你的地图制作工具以处理不完美的声纳信息
- 使用贝叶斯状态估计构建一个强大的地图制作器
- 用一台真正的机器人展示一个完整的映射和规划系统
- 优化您的系统，例如通过改进计划或添加启发式规则

  这些搜索算法，并在一场竞赛中与你的机器人一较高下！

- **签单：**签单应在本周的设计实验课上以及下周的软件实验课上提交。你们应在两个实验课上工作，但实验时间之外不应做任何工作。

- **Windows：**我们正在使用的图形软件有时会在 Windows Vista 和 Windows 7 系统下崩溃。请使用实验室笔记本电脑。

资源：在能够运行 SOAR 的计算机上进行实验。以下是相关文件：

- mapMakerSkeleton.py：用于编写地图制作代码的文件
- mapAndReplanBrain.py：用于运行地图生成器的脑文件
- bayesMapSkeleton.py：用于编写您的贝叶斯地图表示的文件
- robotRaceBrain.py：用于在真实机器人上运行的脑部文件
- mapAndRaceBrain.py：用于在模拟中运行的智能体文件；打印出计时信息
- dl14World.py、bigPlanWorld.py、raceWorld.py、lizWorld.py、mazeWorld.py、mapTestWorld.py：翱翔世界文件

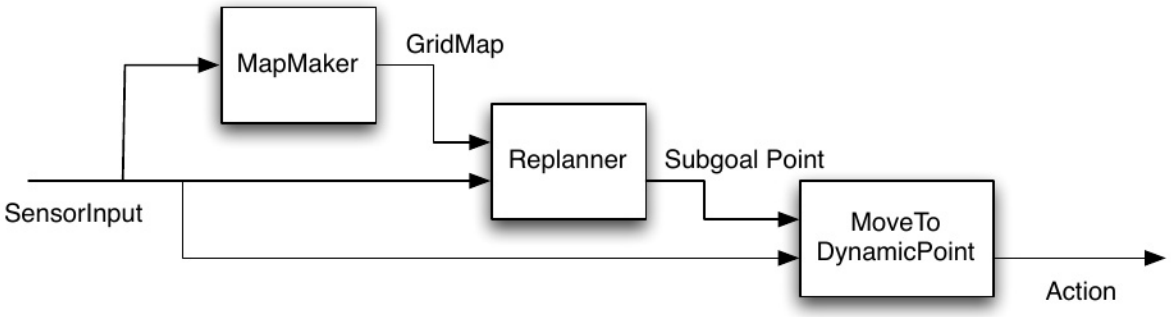
1 引言

在这个实验中，我们将把**软件实验 14** 中的规划器与一个状态机（名为 Map-Maker）连接起来，该状态机会在机器人移动于世界中时动态构建地图。机器人将会，

Some of the software and design labs contain the command athrun 6.01 getFiles. Please disregard this instruction; the same files are available on the 6.01 OCW Scholar site as a .zip file, labeled Code for [Design or Software Lab number].

乐观地设想，一开始就假设它不知道的所有位置都没有障碍物，它将基于此制定计划，然后开始执行该计划。但是，随着它的移动，它会通过声纳看到障碍物，并将其添加到地图上。如果它开始认为当前的计划无法实现，它就会重新规划。因此，在一开始对环境一无所知的情况下，机器人将能够构建一张地图。我们将先构建一个简单的地图生成器，然后看看当传感器数据变得不那么可靠时会发生什么，最后将地图生成器适应处理不可靠的传感器数据。

这是我们将要构建的系统架构的图表。



我们的架构有三个模块。我们将向您提供重新规划器和移动到给定点的模块的实现；您将专注于地图制作器。

状态机的“move.**MoveToDynamicPoint**”类以 **util.Point** 的实例作为输入，并生成 **io.Action** 的实例作为输出。这意味着机器人“瞄准”的点会随着时间的推移动态变化。（请记住，您在 **Design Lab 3** 中编写过这样一台机器！）

重新规划器.ReplannerWithDynamicMap 状态机在初始化时将一个目标点作为参数。目标点是一个 util.Point，指定了机器人世界中位置的最终目标，该目标点将保持固定。机器人的传感器输入（包含机器人当前位置的信息）以及由 MapMaker 输出的 DynamicGridMap 实例将是该机器的输入。重新规划器在第一步制定新计划，将其绘制到地图上，并输出第一个“子目标”（即机器人应该移动到下一个的网格方块的中心），将其作为输入提供给驱动状态机。在后续步骤中，重新规划器执行两件事：

1. 它会检查当前计划中的第一个或第二个子目标位置在世界地图上是否被阻塞。如果是，它会调用规划器制定一个新的计划。
2. 它会检查是否已达成当前的子目标；如果是，它就会从存储的计划前端移除该子目标，并开始将列表中的下一个子目标作为输出生成。

2 制图师，制图师，为我画一张地图

您的任务是在 mapMakerSkeleton.py 文件中编写一个状态机类 MapMaker。它将接受 io.SensorInput 的一个实例作为输入。它的状态将是我们正在创建的地图，该地图可以用 **dynamicGridMap.DynamicGridMap** 的**一个实例**来表示，类似于 basicGridMap.BasicGridMap，但它允许动态构建地图，而不是从文件中创建地图。网格地图既是该机器的状态又是其输出。MapMaker 的初始状态可以是初始的 dynamicGridMap.DynamicGridMap 实例。

出于效率原因，我们将违反我们的状态机协议，并规定您的 get-NextValues 方法应该返回作为旧状态传入的 dynamicGridMap.DynamicGridMap 的*同一个实例*，作为下一个状态和输出。它应该使用 setCell 和 clearCell 方法对该地图进行更改。（在本实验的这一部分，我们不需要 clearCell 方法，但稍后我们会使用它）。如果我们每次都复制它，程序将会极其缓慢。

动态网格地图的 DynamicGridMap 类提供了这些方法：

- __init__(self, xMin, xMax, yMin, yMax, gridSquareSize)： 初始化一个网格，其最小和最大真实世界坐标范围由参数指定，网格方格大小也由参数指定。网格存储在属性grid中。最初，所有值都设置为False，表示它们**未**被占用。
- setCell(self, (ix, iy))： 将索引为 （ix, iy） 的网格单元设置为已占用（即，将存储在该单元中的值设置为 True）。
- clearCell(self, (ix, iy))： 将索引为 （ix, iy） 的网格单元设置为**未**占用（即，将存储在该单元中的值设置为 False）。
- occupied(self, (ix, iy))： 如果索引为 （ix, iy） 的单元被障碍物占据，则返回 True 。
- robotCanOccupy(self, (ix, iy))： 如果机器人将其中心点放置在当前单元格的任何位置都是安全的，则返回 True 。
- squareColor(self, (ix, iy))： 返回应绘制网格单元格 （ix, iy） 的颜色；在这种情况下，如果该单元格被标记为被障碍物占据，则以黑色绘制正方形。如果单元格未被障碍物占据，但由于它们与障碍物单元格太近而机器人无法占据，则以灰色绘制正方形；灰色单元格由 robotCanOccupy 计算得出。

地图制作者应该怎么做？它对世界最基本的信息是，在声纳射线末端的网格单元被障碍物占据。所以，在每一步，对于每个声纳传感器，如果其值小于 **sonarDist.sonarMax**，则应将地图中包含声纳射线末端点的网格单元标记为包含障碍物。您在辅导问题 **Wk.11.1.6** 中编写的 sonarHit 过程可用作

```
sonarDist.sonarHit(dist, sonarPose, robotPose)
```

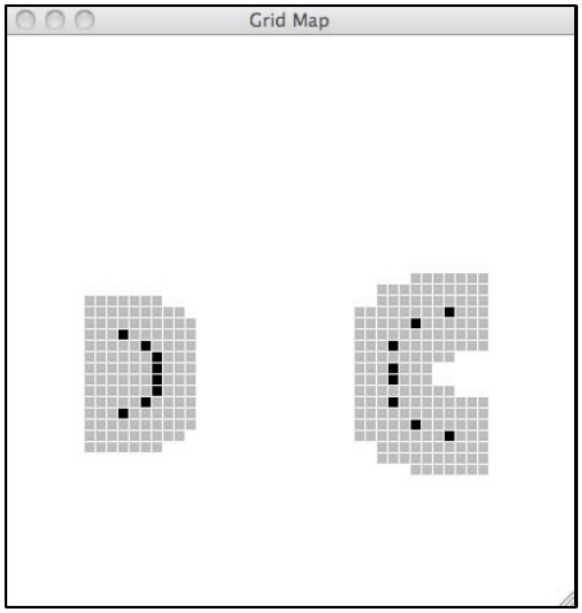
相对于机器人，所有声纳传感器的姿势列表可在 sonarDist.sonarPoses 中获取。

第一步。

自我检查 1. 为了在空闲状态下进行测试，我们定义了一个 SensorInput 类来模拟 soar 中的 io. SensorInput 类。考虑这两个可能的传感器输入实例（每个实例都包含一个由 8 个实数值的声纳读数和一个位姿组成的列表）。

```
testData = [SensorInput([0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2],
                        util.Pose(1.0, 2.0, 0.0)),
            SensorInput([0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4, 0.4],
                        util.Pose(4.0, 2.0, -math.pi))]
```

一定要理解它们为何会导致如下所示的地图。记住，黑色方块是唯一被标记为已占用的方块，这是由于声纳读数；灰色方块是机器人无法占据的位置（因为它会与其中一个黑色位置发生碰撞）。



步骤 2. 实现 MapMaker 类。它将被如下调用：

MapMaker（x 的最小值、x 的最大值、y 的最小值、y 的最大值、网格方格大小）

记住要初始化 startState 属性，并定义一个 getNextValues 方法，用于标记输入中声纳射线末端的单元格。

第三步。在 idle 中测试您的地图生成器（务必使用 -n 标志启动）。通过以下操作进行：

testMapMaker（测试数据）

它将创建一个您的 MapMaker 类的实例，并使用“自我检查”问题中的测试数据对其调用 transduce 函数。验证您的结果与图中的结果是否一致。

第 4 步。现在，在 SOAR 中测试您的代码。文件 mapAndReplanBrain.py 包含了将系统的所有部分连接在一起形成一个能在 SOAR 中运行的大脑所需的组合状态机。

您可以在大脑文件顶部描述的任何世界中工作；在 SOAR 中选择相应的模拟世界，然后确保您有一行类似的内容

使用世界（dl14World）

使用与您正在使用的世界相对应的名称，该名称会为您正在处理的世界选择适当的维度（选择 dl14World 的这一行目前在大脑中）。在测试代码时，请务必使用与您所选世界文件相对应的模拟世界。

一个窗口会弹出来，显示地图和计划的当前状态。黑色的方块是地图制作者标记为已占用的方块。有时方块会以灰色显示：这意味着，尽管它们没有被障碍物占用，但机器人无法占用。并不是所有不可占用的方块都会以灰色显示（我们不想频繁重绘整个屏幕）。

核对 1。 **第 14 周 2.1**：向一名工作人员展示您的地图制作者绘制的地图。如果它有任何令人惊讶之处，请解释原因。动态更新的地图如何与规划和重新规划过程相互作用？

3 嘈杂的噪音惹恼了一只牡蛎

第 5 步。默认情况下，在 soar 中的声纳读数是完美的。但在真正的机器人中，声纳读数绝非完美。在 mapAndReplanBrain.py 中查找这样一行：

```
soar.outputs.simulator.SONAR_VARIANCE = lambda mean: noNoise
```

并将其改为其中之一

```
soar.outputs.simulator.SONAR_VARIANCE = lambda mean: smallNoise
soar.outputs.simulator.SONAR_VARIANCE = lambda mean: mediumNoise
```

这会将声纳噪声模型的默认方差（高斯分布的宽度）增加到一个非零值。

自我检查 2. 让大脑在这些更嘈杂的世界中再次运行。它为什么不起作用？传感器读数中的噪声如何影响其性能？

事实上，我们从声纳传感器获取的信息不仅仅是射线末端是否被占用这一事实。我们还知道，沿着声纳射线，从传感器到最末一个网格单元之间的网格单元都是空的。即使声纳读数大于最大良好值，您也可能考虑将射线第一部分沿线的单元标记为空的。

第 6 步。改进你的 MapMaker 类以利用此信息。您可能会发现 **util.lineIndices（start, end）** 这个程序很有用：start 和 end 各自应该是一对

（x, y） 整数网格单元*索引*；返回值是一个*由* （x, y） 整数网格单元索引对*组成的*列表,*这些*对构成*了*起点和终点*之间的*线段（包括*起点*和终点）。您可以*将*这些单元视为根据声纳测量可以合理地标记*为空闲的*网格位置*的*集合。*务必不要清除最后一个点,即您已经标记为已占用的那个点;尽管现在可能行得通,但如果您每次清除然后标记该单元,这将在第 4 节中引起问题,届时我们将使用状态估计器来聚合随时间推移关于每个网格单元获得的证据。*

第 7 步。通过在 Idle 中操作来测试您新的 MapMaker 。

```
testMapMakerClear(testClearData)
```

请注意，这是 testMapMakerClear，它与 testMapMaker 是不同的程序。它将创建您的地图制作器的实例，并将所有网格方块初始设置为已占用状态。然后，它将使用此输入调用 transduce：

```
testClearData = [SensorInput([1.0, 5.0, 5.0, 1.0, 1.0, 5.0, 5.0, 1.0],
                             util.Pose(1.0, 2.0, 0.0)),
                 SensorInput([1.0, 5.0, 5.0, 1.0, 1.0, 5.0, 5.0, 1.0],
                             util.Pose(4.0, 2.0, -math.pi))]
```

*自我*检查 *3.* 预测生成的地图应该是什么样子，并确保你的代码能生成正确的内容。

第 8 步。再次在 SOAR 中运行 mapAndReplanBrain 并确保你理解在没有噪声和中度噪声的情况下所发生的情况。

检查 2. 第 14 周 **2.2**：展示您新的地图生成器运行的情况，先是没有噪音，然后是有中等噪音。我们不一定期望它能可靠运行：但您应该解释它在做什么以及为什么。

4 贝叶斯图

第 3 次和第 4 次签到应在软件实验室 15 进行，但我们强烈鼓励您留在设计实验室 14 并尝试完成它。如果您在设计实验室期间通过了签到，那么您就不需要参加软件实验室 15 了。

在存在噪声的情况下，使映射更可靠的一种方法是将此问题视为*状态估计*问题：我们对网格单元的底层状态有不可靠的观测值，并且我们可以随着时间的推移聚合这些信息。

对于状态估计，我们可能的假设空间应该是所有可能的映射空间。但如果我们的映射是一个 20×20 的网格，那么可能的网格映射数量就是 2^{400}（每个单元格要么 ）

是否被占用，这就像 400 位二进制数字的编号一样），这是一个*过于*庞大的空间，难以进行估计。为了使这个问题在计算上易于处理，我们将做出一个非常强烈的**独立性假设：地图每个**方格的**状态**与**其他方格**的**状态无关**。如果我们这样做，那么，我们不是有一个具有 2 - λ 个状态的单个状态估计问题，而是有 400 个状态估计问题，每个问题都有 2 个状态（网格单元要么被占用，要么未被占用）。

幸运的是，我们已经为状态估计构建了一个不错的状态机类，并且我们可以用它来构建动态网格地图（DynamicGridMap）的一个新的子类，在这个子类中，网格中的每个单元都包含一个 **seFast.StateEstimator** 的**实例**（这是您在 **Wk.11.1.3** 中实现的）。

您的任务是在“bayesMapSkeleton.py”文件中为“BayesGridMap”类编写定义。在此之前，您需要思考如何将状态估计器用作网格的元素。

回想一下，seFast.StateEstimator 的 __init__ 方法的参数是 ssm.**StochasticSM** 的**一个实例**，它指定了环境的动态特性。以下是指定单个地图网格单元的世界动态特性时需要考虑的一些要点：

· 该单元有两种可能的状态：被占用或未被占用。

对于这个单元，我们可能会做出两种可能的观察结果：它是空的，或者它曾是声纳命中点的位置。

· 您可以假设环境是完全静态的：也就是说，网格单元的实际状态永远不会改变，尽管随着您收集观察结果，您对其的看法会发生变化。但是，如果您愿意，您也可以考虑环境发生变化的情况，也许是因为家具被移动了。

> ***自我检查 4.* 记住，声纳波束有时会从障碍物上反弹回来。**
> 不返回传感器，并且当我们说一个正方形是清晰的，我们指的是
> 它里面任何地方都没有东西。你觉得我们（.....）的可能性有多大?
> 当观察一个细胞时，发现它实际上是被占据的，却被误认为是空闲的？这就是我们所观察到的。
> 在它真的无人使用时出击？那么先验（初始）概率应该是多少?
> bilities 是指任何特定细胞被占据的情况吗?
> 确定细胞状态的可能值。假设观察者
> “vasion”可以是“hit”，如果该单元有声纳命中；也可以是“free”，如果
> 声纳穿过牢房。为**避免混淆，给它们取些名字**。
> **既非“命中”也非“未命中”的内部状态**。
> 如果您在制定初始分布、观测方面遇到困难
> 并且对于状态估计器的转换模型，与工作人员交流。

第 9 步。在 bayesMapSkeleton.py 中编写代码，创建一个 ssm.StochasticSM 的实例，用于对**单个网格单元**的行为进行建模。

第 10 步。通过以下操作测试您的网格单元模型

测试细胞动力学（cellSSM，您的测试输入）

其中 cellSSM 是 ssm.StochasticSM 的一个实例，并且您的测试输入是以下列表之一。它将为单个网格单元创建一个状态估计器的实例，并向其提供一系列观测值。然后，它将使用数据输入调用 transduce 。

如果您给它提供此输入数据，它对于该单元被占用的最终置信度是多少？（为什么这里都是 None？）

```
mostlyHits = [('hit', None), ('hit', None), ('hit', None), ('free', None)]
```

如果您给它这个输入数据怎么样?

```
mostlyFree = [('free', None), ('free', None), ('free', None), ('hit', None)]
```

第 11 步。现在是**思考**实现 BayesGridMap 类策略的时候了；先不要开始实现。

您将不得不自行管理每个单元中状态估计器机器的初始化和状态更新。您应该确保在创建此网格后，立即在每个状态估计器状态机器上调用 start 方法。每当您获得有关单元状态的新证据时，您还必须调用估计器的 step 方法，输入为（o，a），其中 o 是观测值，a 是动作；实际上，在这个模型中，我们将忽略动作参数，所以您可以将 None 作为 a 的输入。

您可以通过查看在线软件文档来提醒自己创建状态估计器以及启动和推进状态机的正确方法。一旦状态机 mysm 已启动，您可以使用 mysm.state 访问其内部状态。

您的贝叶斯网格地图将是**动态网格地图**的一个子类，并且可以直接基于 DynamicGridMap.py 的以下方面进行建模：

```
class DynamicGridMap(gridMap.GridMap):
    def makeStartingGrid(self):
        return util.make2DArray(self.xN, self.yN, False)
    def squareColor(self, (xIndex, yIndex)):
        if self.occupied((xIndex, yIndex)): return 'black'
        else: return 'white'
    def setCell(self, (xIndex, yIndex)):
        self.grid[xIndex][yIndex] = True
        self.drawSquare((xIndex, yIndex))
    def clearCell(self, (xIndex, yIndex)):
        self.grid[xIndex][yIndex] = False
        self.drawSquare((xIndex, yIndex))
    def occupied(self, (xIndex, yIndex)):
        return self.grid[xIndex][yIndex]
```

第 12 步

> **第 14 周 2.3** 解决这个关于对象实例集合的辅导问题。

第 13 步。这里对您需要编写的方法有一些进一步的描述。请记住，动态网格地图中的值网格存储在“grid”属性中。我们不需要编写“__init__”方法，因为它是从“DynamicGridMap”继承而来的。

- makeStartingGrid(self)：构建并返回二维数组（列表的列表），其中包含 seFast.StateEstimator 的实例。您可以使用 self 的 xN 和 yN 属性来确定数组的大小。您应该使用 **util.make2DArrayFill** 来实现（请确保您明白为什么 make2DArray 不适用）。

- setCell(self, (xIndex, yIndex))： 对于在本单元格中观测到有声纳命中这一情况，此方法应在本单元格的状态机上执行状态机更新。并且，如果单元格的颜色发生了变化，它应该重新绘制地图中的正方形。
- clearCell(self, (xIndex, yIndex))： 对于观察到该单元格为空闲的情况，此方法应在该单元格的状态机上执行状态机更新。并且，如果单元格的颜色发生了变化，它应该重新绘制地图中的正方形。
- occProb(self, (xIndex, yIndex))： 此方法返回一个介于 0 到 1 之间的浮点数，表示我们认为指定单元格被占用的概率。这被 squareColor 方法用于显示目的，该方法已经编写好了。
- occupied(self, (xIndex, yIndex))： 如果出于规划目的应将该单元格视为已占用，则此方法返回 True；否则返回 False。您可能得对此方法稍作试验，以找到判断该方格已占用的概率的良好阈值。使用上述指定的 occProb 方法。

第 14 步。现在，在 bayesMapSkeleton.py 中实现 BayesGridMap 类。它已经定义了 square-Color 方法。

第 15 步。通过以下方式在 Idle 中测试您的代码:

- 将您的 MapMaker 更改为使用 bayesMap.BayesGridMap 而非 dynamicGridMap.DynamicGridMap。无需对该类进行进一步更改。
- 在 Idle 中运行 mapMakerSkeleton.py ，然后在 shell 中输入:

```
testMapMakerN(1, testData)
```

它将使用与动态网格地图相同的数据进行更新。现在，弹出的窗口使用了不同的配色方案：白色表示可能畅通，亮绿色表示可能受阻，颜色之间有连续的变化。如果一个单元格被视为受阻，则将其涂成黑色；如果一个单元格未受阻，但机器人也无法占用，则将其涂成红色。

如果您输入

```
testMapMakerN(2, testData)
```

然后它将使用给定的数据更新地图两次。

> *自我检查 5.* 尝试进行两次更新。尝试使用 testClearData。确保这一切都合情合理。

第 16 步。现在，像之前一样在 soar 中测试您的映射器，运行 mapAndReplanBrain 。您可能会发现使用步骤按钮特别有用。

检查 3. **第 14 周 2.4**：使用带有中等噪声和高噪声的贝叶斯地图模块在 mapAndReplanBrain 中展示您的地图绘制器。如果它在高噪声下无法运行，请解释问题所在，但您不一定需要让它在高噪声下运行。

4.1 真实机器人

现在，让我们看看这在现实世界中效果如何！把笔记本电脑带到现实世界的游乐场之一，将其连接到机器人上，并运行 robotRaceBrain.py。您可能得调整状态估计器中的参数（通常是误报率或考虑一个正方形被阻塞的阈值），以便它能够可靠地工作。

核对 4。 **第 14 周 2.5**：在真正的机器人上展示你的绘图仪。你可以在游戏围栏里移动障碍物以增加乐趣，但要确保不要设置得让机器人无法从起点到达目标。

5 可选：出发，速度赛车手，出发！

在**软件实验室 14** 中，我们通过使用**带有启发式**的 **A*搜索**来加快规划时间。并且我们的机器人可以通过构建地图和规划路径来避开障碍物。现在，我们要致力于让我们的机器人在世界中移动得更快。在这个实验室期间，你的任务是尽可能加快你的机器人；最后，我们将进行一场竞赛。

在本节中，我们将重点关注重新规划器和移动到动态点模块。

mapAndRaceBrain.py 目前被设置为在 raceWorld.py 中运行：选择 raceWorld.py 作为模拟的世界，并运行大脑。若要更改您正在处理的世界，请更改大脑中的 useWorld 行（并且记得更改模拟的世界）。

当您使用 mapAndRaceBrain.py 运行时，您会注意到当机器人到达目标时，它会停下来并打印出类似的内容

```
Total steps: 320
Elapsed time in seconds: 209.554840088
```

这是执行您的计划所采取的原始步骤的数量，以及所花费的经过时间。这些数字将是您的“分数”。请注意，我们的目标是低分数！

您可以在自己的笔记本电脑或实验室的笔记本电脑上进行调试，但只有在实验室的笔记本电脑上运行得出的分数才会被视为正式成绩。

第 17 步

自我测试 6. 让您的机器人在 raceWorld 中运行，看看您能得到多少分数。把这个分数写下来，因为这是您进步的基准。

请注意，在机器人执行其计划的过程中，有几种情况会使其减速：

- 从网格单元到网格单元的每一个单独步骤，都由 move.MoveToFixedPoint 中的比例控制器控制。该控制器必须减速，以小心地达到每个子目标。
- 旋转需要很长时间。

以下是一些解决这些问题的可能策略。您不必全部采用或采用所有策略。如果您自行选择了一种（我们鼓励这样做！），请与工作人员交流。

您可以通过制定一个需要较少停车和/或较少转向的计划来加快机器人的速度。通过编辑您的 GridDynamics 类（您可以从文件 swLab14/plannerStandaloneSkeleton.py 中将其复制到 replannerRace.py 中）或 replannerRace.py 中的 ReplannerWith-DynamicMap 类来实现这些（请仔细阅读该代码）。

1. 按照最初的一系列动作进行规划，但随后对规划进行后处理，以使其更高效。如果规划要求机器人沿直线移动数次，您可以放心地删除中间的子目标，直到机器人最终必须转弯的位置。
2. 扩大你计划纳入机器人航向的空间。对导致机器人旋转的动作施加额外的惩罚。尝试调整惩罚力度以提高你的得分。（这很难做到正确；只有在你有大量空闲时间的情况下才去做）
3. 增加动作集合，将距离超过一个单位的移动包括在内。您可以使用 **util.lineIndicesConservative（(ix1, iy1), (ix2, iy2)）** 程序获取一个列表，其中包含机器人从 （ix1, iy1） 出发到 （ix2, iy2） 结束所需遍历的网格单元。这个网格单元列表是保守的，因为它不会走任何捷径。

您还可以通过更改 move.**MoveToDynamicPoint** 行为（在 move.py 中）中的增益和容差来加快路径的执行。阅读文件中的代码以了解这些参数的含义，然后考虑对其进行调整以改善机器人的行为。但请确保您不会导致机器人撞到障碍物！您可以在 mapAndRace-Brain.py 中编辑这些代码行。

```
move.MoveToFixedPoint.forwardGain = 1.0
move.MoveToFixedPoint.rotationGain = 1.0
move.MoveToFixedPoint.angleEps = 0.05
```

不允许更改 maxVel 参数。

第 18 步。实施一些改进措施以使机器人走得更快。

Check Yourself 7. Post your best scores in simulation on `raceWorld` and `lizWorld` on the board.

第 19 步。使用 robotRaceBrain.py 在真正的机器人上运行。它有一对良好的起点和终点值，以及房间前方大世界大小的边界。它只能在机器人上运行。如果您想在模拟环境中进行测试，可以切换回使用 mapAndRaceBrain.py 。

检查 5. **第 14 周 2.6**：将您在真实机器人竞赛中的最佳得分发布在黑板上，同时也要发布在辅导问题**第 14 周 2.6** 中。获胜者将获得特别奖品！

6.01SC 《电气工程与计算机科学导论》 2011 年春季